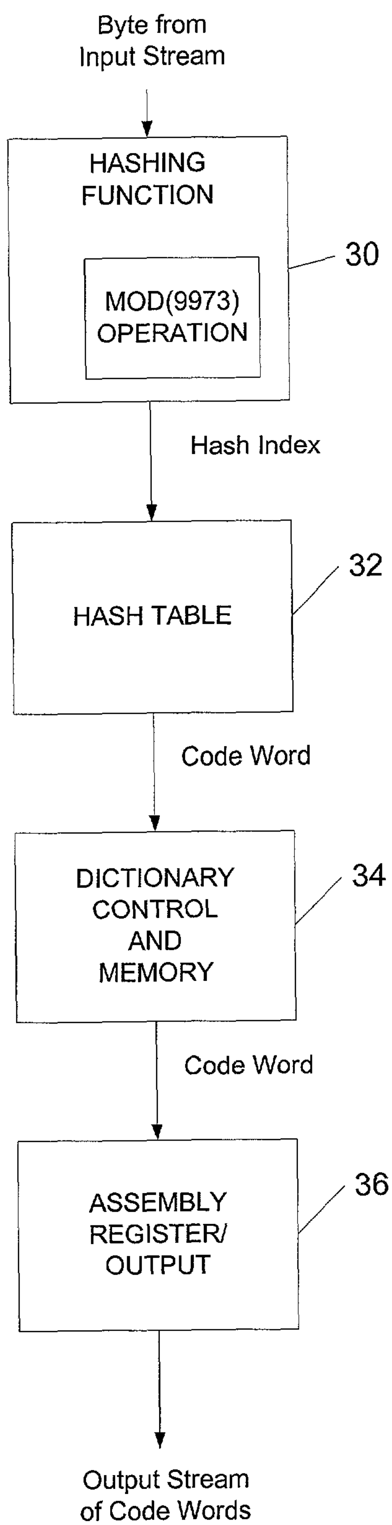

Byte from Input Stream
HASHING FUNCTION
MOD(9973) OPERATION
30
Hash Index
HASH TABLE
32
Code Word
DICTIONARY CONTROL AND MEMORY
34
Code Word
ASSEMBLY REGISTER/ OUTPUT
36
Output Stream of Code Words

FIGURE 1

FIGURE 2

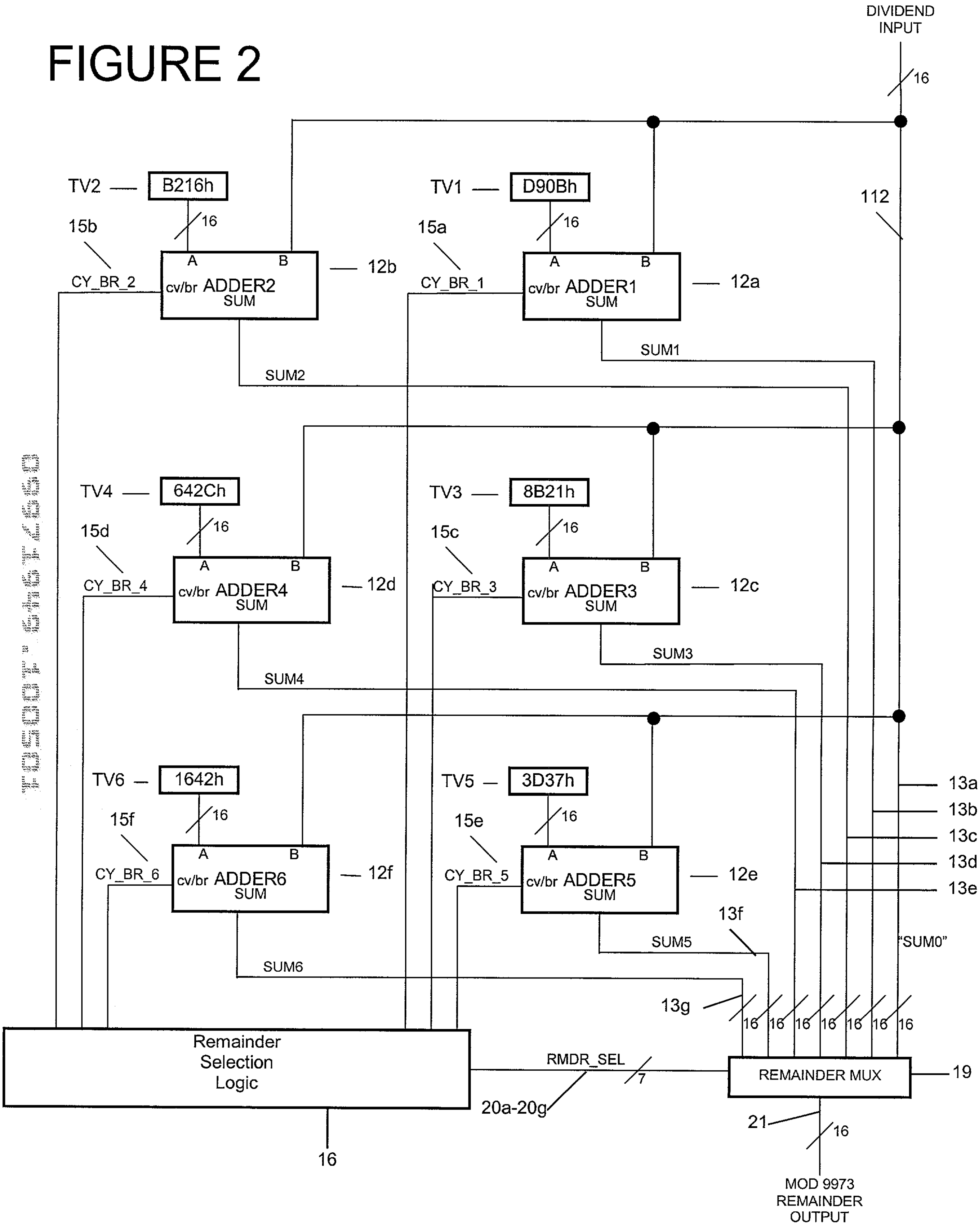
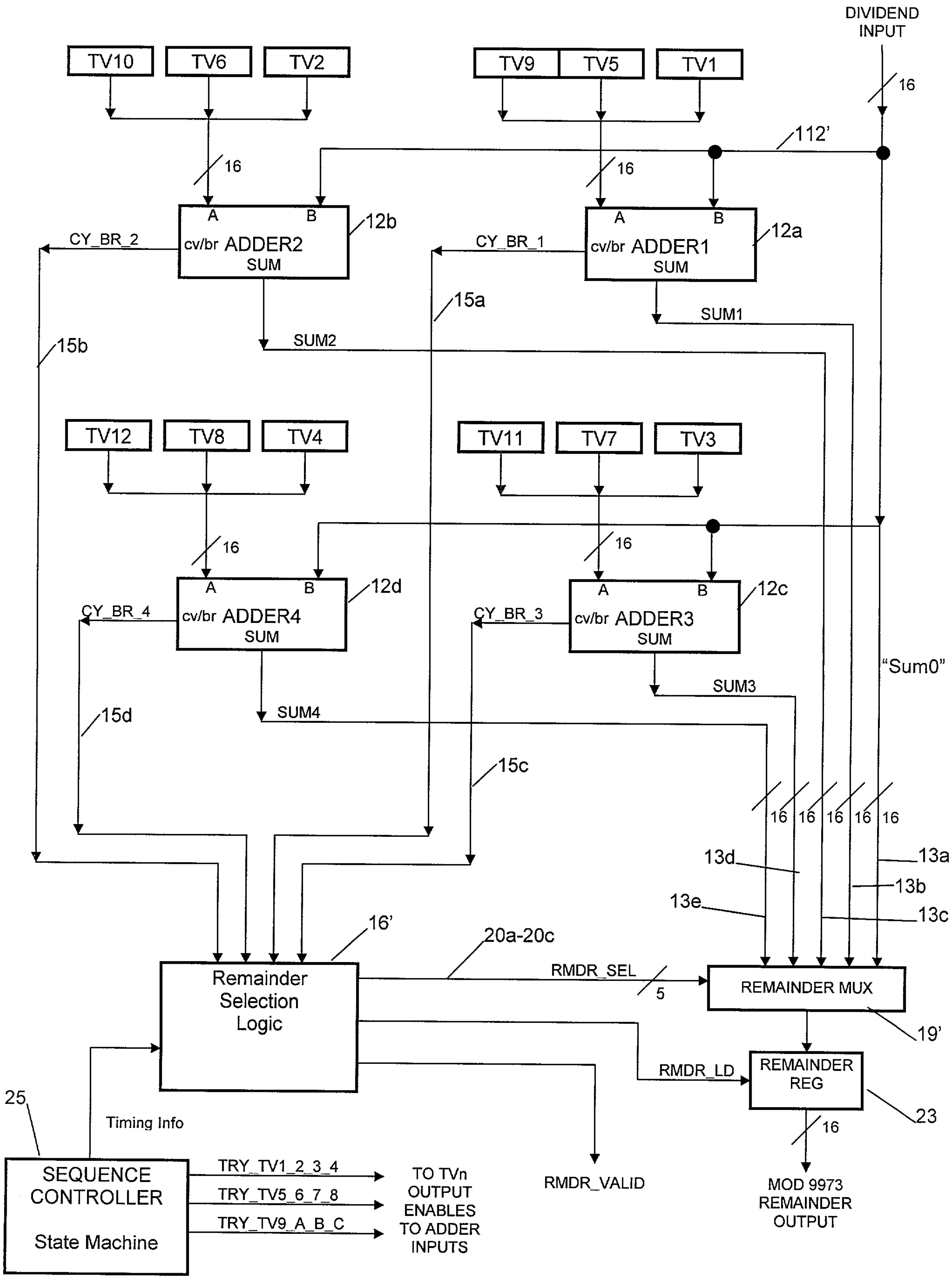

DIVIDEND INPUT
16
112
TV2
B216h
TV1
D90Bh
15b
15a
CY_BR_2
CY_BR_1
A
B
cv/br ADDER2
SUM
cv/br ADDER1
SUM
12b
12a
SUM1
SUM2
TV4
642Ch
TV3
8B21h
15d
15c
CY_BR_4
CY_BR_3
cv/br ADDER4
SUM
cv/br ADDER3
SUM
12d
12c
SUM3
SUM4
TV6
1642h
TV5
3D37h
15f
15e
CY_BR_6
CY_BR_5
cv/br ADDER6
SUM
cv/br ADDER5
SUM
12f
12e
13a
13b
13c
13d
13e
13f
"SUM0"
SUM5
SUM6
13g
Remainder Selection Logic
RMDR_SEL
7
REMAINDER MUX
19
20a-20g
21
16
MOD 9973 REMAINDER OUTPUT

FIGURE 2A


DIVIDEND INPUT
TV10
TV6
TV2
TV9
TV5
TV1
16
112'
A
B
cv/br ADDER2
SUM
12b
cv/br ADDER1
SUM
12a
CY_BR_2
CY_BR_1
15a
15b
SUM1
SUM2
TV12
TV8
TV4
TV11
TV7
TV3
cv/br ADDER4
SUM
12d
cv/br ADDER3
SUM
12c
CY_BR_4
CY_BR_3
"Sum0"
SUM3
SUM4
15d
15c
13a
13b
13c
13d
13e
16'
20a-20c
Remainder Selection Logic
RMDR_SEL
5
REMAINDER MUX
19'
RMDR_LD
REMAINDER REG
23
25
Timing Info
SEQUENCE CONTROLLER
State Machine
TRY_TV1_2_3_4
TRY_TV5_6_7_8
TRY_TV9_A_B_C
TO TVn OUTPUT ENABLES TO ADDER INPUTS
RMDR_VALID
MOD 9973 REMAINDER OUTPUT

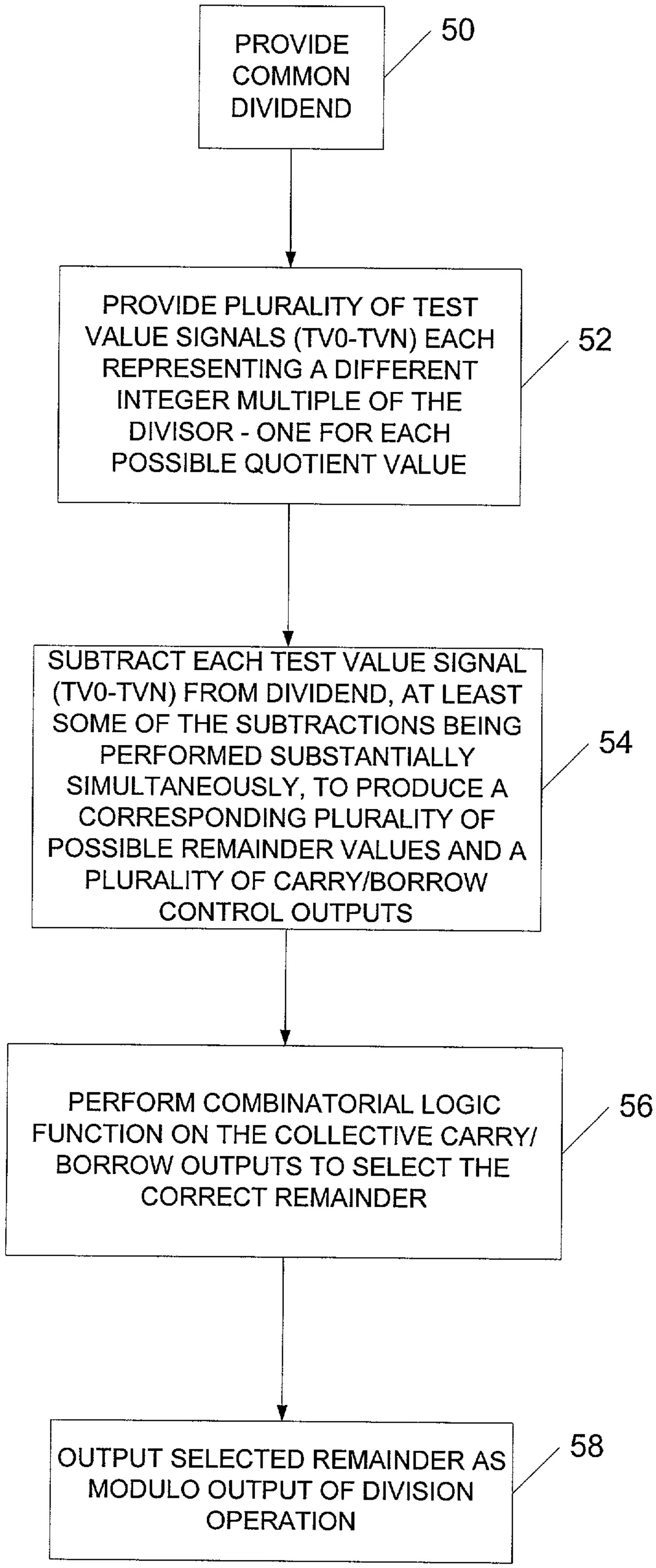

PROVIDE COMMON DIVIDEND
50
PROVIDE PLURALITY OF TEST VALUE SIGNALS (TV0-TVN) EACH REPRESENTING A DIFFERENT INTEGER MULTIPLE OF THE DIVISOR - ONE FOR EACH POSSIBLE QUOTIENT VALUE
52
SUBTRACT EACH TEST VALUE SIGNAL (TV0-TVN) FROM DIVIDEND, AT LEAST SOME OF THE SUBTRACTIONS BEING PERFORMED SUBSTANTIALLY SIMULTANEOUSLY, TO PRODUCE A CORRESPONDING PLURALITY OF POSSIBLE REMAINDER VALUES AND A PLURALITY OF CARRY/BORROW CONTROL OUTPUTS
54
PERFORM COMBINATORIAL LOGIC FUNCTION ON THE COLLECTIVE CARRY/BORROW OUTPUTS TO SELECT THE CORRECT REMAINDER
56
OUTPUT SELECTED REMAINDER AS MODULO OUTPUT OF DIVISION OPERATION
58

FIGURE 3

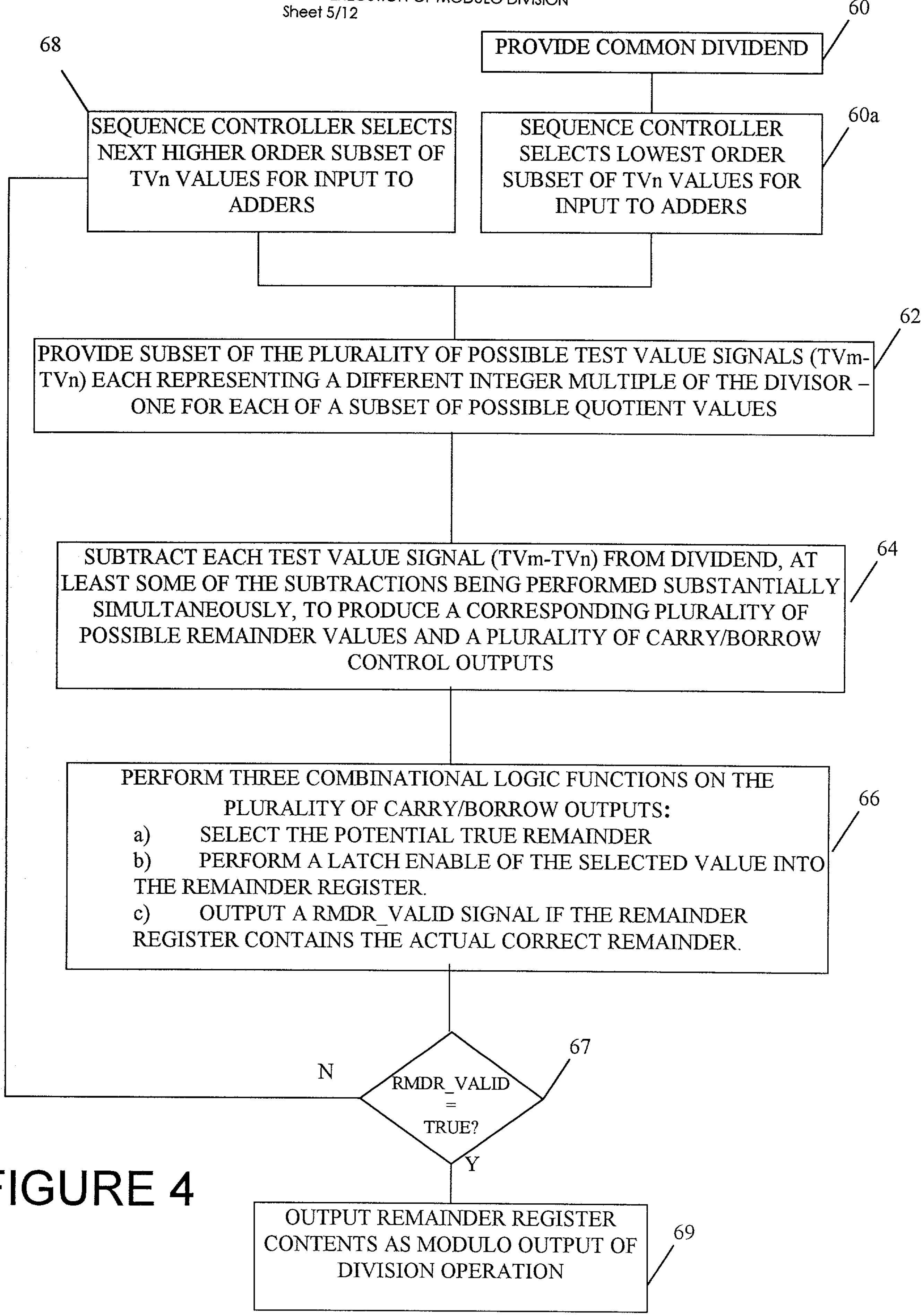

PROVIDE COMMON DIVIDEND
60
SEQUENCE CONTROLLER SELECTS LOWEST ORDER SUBSET OF TVn VALUES FOR INPUT TO ADDERS
60a
SEQUENCE CONTROLLER SELECTS NEXT HIGHER ORDER SUBSET OF TVn VALUES FOR INPUT TO ADDERS
68
PROVIDE SUBSET OF THE PLURALITY OF POSSIBLE TEST VALUE SIGNALS (TVm-TVn) EACH REPRESENTING A DIFFERENT INTEGER MULTIPLE OF THE DIVISOR – ONE FOR EACH OF A SUBSET OF POSSIBLE QUOTIENT VALUES
62
SUBTRACT EACH TEST VALUE SIGNAL (TVm-TVn) FROM DIVIDEND, AT LEAST SOME OF THE SUBTRACTIONS BEING PERFORMED SUBSTANTIALLY SIMULTANEOUSLY, TO PRODUCE A CORRESPONDING PLURALITY OF POSSIBLE REMAINDER VALUES AND A PLURALITY OF CARRY/BORROW CONTROL OUTPUTS
64
PERFORM THREE COMBINATIONAL LOGIC FUNCTIONS ON THE PLURALITY OF CARRY/BORROW OUTPUTS:
a) SELECT THE POTENTIAL TRUE REMAINDER
b) PERFORM A LATCH ENABLE OF THE SELECTED VALUE INTO THE REMAINDER REGISTER.
c) OUTPUT A RMDR_VALID SIGNAL IF THE REMAINDER REGISTER CONTAINS THE ACTUAL CORRECT REMAINDER.
66
RMDR_VALID = TRUE?
67
N
Y
OUTPUT REMAINDER REGISTER CONTENTS AS MODULO OUTPUT OF DIVISION OPERATION
69

FIGURE 4

All values shown are decimal except "K" is 2^10 (1024 decimal) multiplier.

2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K 2K

64K (16 bits)

Range Width	9973	9973	9973	9973	9973	9973	5698
DIVIDEND	0000 … 9972	9973 … 19945	19946 … 29918	29919 … 39891	39892 … 49864	49865 … 59837	59838 … 65535
MOD Remainder Output	0 … 9972	0 … 9972	0 … 9972	0 … 9972	0 … 9972	0 … 9972	0 … 5697
Quotient	Q=0	Q=1	Q=2	Q=3	Q=4	Q=5	Q=6
Cy/Br, "Adder 0"	- - - - - - - - - -	- - - - - - - - - -	- - - - - - - - - -	- - - - - - - - - -	- - - - - - - - - -	- - - - - - - - -	- - - - -
Cy/Br, Adder 1, "15a"	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1
Cy/Br, Adder 2, "15b"	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1
Cy/Br, Adder 3, "15c"	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1
Cy/Br, Adder 4, "15d"	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1	1 1 1 1 1
Cy/Br, Adder 5, "15e"	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1	1 1 1 1 1
Cy/Br, Adder 6, "15f"	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	1 1 1 1 1
Sel "Sum 0"	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0
Sel Sum 1	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0
Sel Sum 2	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0
Sel Sum 3	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0
Sel Sum 4	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0	0 0 0 0 0
Sel Sum 5	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1	0 0 0 0 0
Sel Sum 6	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	1 1 1 1 1

FIGURE 5

FIGURE 6A

```
--Selects which of the adder output values is the Remainder Output.
   Signal RMDR_SEL : std_logic_vector( 6 downto 0); -- 7 bit vector of signals used
        -- to select the true remainder from the plurality of remainders.
        -- The vector is "one hot" encoded.
   Signal CY_BR_VEC: std_logic_vector(5 downto 0);  -- Vector of CY_BR1...6 bits.
        -- composed of the plurality of carry/borrow outputs of the adders.
   Signal QUOTIENT_VAL: std_logic_vector(2 downto 0);  -- Numerical value of quotient.

-- Adder 0 was reduced out of the design and thus eliminated, being replaced with
--    a straight-through bus, leaving the other 6 adders physically implemented.
--  Create a vector from the adder CY_BR outputs so they can be referenced as a group
--  in the CASE statement in the RMDR_SEL_LOGIC process, below.
  CY_BR_VEC(0) <= CY_BR1; -- CY_BR output of adder 1 is bit 0 of CY_BR_VEC.
  CY_BR_VEC(1) <= CY_BR2; -- CY_BR output of adder 2 is bit 1 of CY_BR_VEC.
  CY_BR_VEC(2) <= CY_BR3; -- CY_BR output of adder 3 is bit 2 of CY_BR_VEC.
  CY_BR_VEC(3) <= CY_BR4; -- CY_BR output of adder 4 is bit 3 of CY_BR_VEC.
  CY_BR_VEC(4) <= CY_BR5; -- CY_BR output of adder 5 is bit 4 of CY_BR_VEC.
  CY_BR_VEC(5) <= CY_BR6; -- CY_BR output of adder 6 is bit 5 of CY_BR_VEC.

------------
--Remainder Select control output to control Remainder Mux, item # 19.
RMDR_SEL_LOGIC : process (CY_BR_VEC)

--RMDR_SEL output signal names and values.  "One-hot" encoded.
Constant ZERO : std_logic_vector(6 downto 0) := "0000001";  -- Select "adder 0" output.
Constant ONE  : std_logic_vector(6 downto 0) := "0000010";  -- Select adder 1 output.
Constant TWO  : std_logic_vector(6 downto 0) := "0000100";  -- Select adder 2 output.
Constant THREE: std_logic_vector(6 downto 0) := "0001000";  -- Select adder 3 output.
Constant FOUR : std_logic_vector(6 downto 0) := "0010000";  -- Select adder 4 output.
Constant FIVE : std_logic_vector(6 downto 0) := "0100000";  -- Select adder 5 output.
Constant SIX  : std_logic_vector(6 downto 0) := "1000000";  -- Select adder 6 output.

-- CY_BR_VEC names and values which match associated numerical quotient values.
Constant QUOT_0 : std_logic_vector(5 downto 0) := "000000";  -- Quotient of dividend is 0.
Constant QUOT_1 : std_logic_vector(5 downto 0) := "000001";  -- Quotient of dividend is 1.
Constant QUOT_2 : std_logic_vector(5 downto 0) := "000011";  -- Quotient of dividend is 2.
Constant QUOT_3 : std_logic_vector(5 downto 0) := "000111";  -- Quotient of dividend is 3.
Constant QUOT_4 : std_logic_vector(5 downto 0) := "001111";  -- Quotient of dividend is 4.
Constant QUOT_5 : std_logic_vector(5 downto 0) := "011111";  -- Quotient of dividend is 5.
Constant QUOT_6 : std_logic_vector(5 downto 0) := "111111";  -- Quotient of dividend is 6.
```

FIGURE 6B

```
-- Perform selection function.
-- Note: Interpretation of the CASE statement.  Example: "WHEN QUOT_0 =>" means, "When the
-- value of CY_BR_VEC equals the value assigned to the constant, QUOT_0, execute the
-- following statements until the next "WHEN..." statement".  Then exit the CASE.

Case CY_BR_VEC is
   When QUOT_0 =>   -- Quotient of dividend is 0.  "Adder 0" has correct mod output value.
      RMDR_SEL     <= ZERO;         -- Select remainder 0.
      QUOTIENT_VAL <= "000"  -- Quotient value is 0 (binary).
   When QUOT_1 =>   -- Quotient of dividend is 1.  Adder 1 has correct mod output value.
      RMDR_SEL     <= ONE;         -- Select remainder 1.
      QUOTIENT_VAL <= "001"  -- Quotient value is 1 (binary).
   When QUOT_2 =>   -- Quotient of dividend is 2.  Adder 2 has correct mod output value.
      RMDR_SEL     <= TWO;         -- Select remainder 2.
      QUOTIENT_VAL <= "010"  -- Quotient value is 2 (binary).
   When QUOT_3 =>   -- Quotient of dividend is 3.  Adder 3 has correct mod output value.
      RMDR_SEL     <= THREE;         -- Select remainder 3.
      QUOTIENT_VAL <= "011"  -- Quotient value is 3 (binary).
   When QUOT_4 =>   -- Quotient of dividend is 4.  Adder 4 has correct mod output value.
      RMDR_SEL     <= FOUR;         -- Select remainder 4.
      QUOTIENT_VAL <= "100"  -- Quotient value is 4 (binary).
   When QUOT_5 =>   -- Quotient of dividend is 5.  Adder 5 has correct mod output value.
      RMDR_SEL     <= FIVE;         -- Select remainder 5.
      QUOTIENT_VAL <= "101"  -- Quotient value is 5 (binary).
   When QUOT_6 =>   -- Quotient of dividend is 6.  Adder 6 has correct mod output value.
      RMDR_SEL     <= SIX;         -- Select remainder 6.
      QUOTIENT_VAL <= "110"  -- Quotient value is 6 (binary).

End case;
```

FIGURE 7A

Note: Interpretation of VHDL:

a) CASE statement. Example:

   Case SEQ_STATE is

   WHEN CYCLE_1 then

   "**WHEN CYCLE_1**" means, "When the value of SEQ_STATE equals the value assigned to the constant, CYCLE_1 , execute the statements following the WHEN until the next "WHEN..." statement". Then exit the CASE.

b) The symbol "**<=**" is interpreted as: "is assigned the value of...".

```
****************** BEGINNING OF "VHDL" DESCRIPTION ************************
--Signal declarations and definitions.
   Signal CY_BR_VEC: std_logic_vector(3 downto 0);  -- 4 bit vector of CY_BR signals.
        -- composed of the plurality of carry/borrow outputs of the adders.
   Signal QUOTIENT_VAL: std_logic_vector(3 downto 0); -- 4 bit numeric value of quotient.
   Signal SEQ_STATE  : std_logic_vector(1 downto 0);  -- Sequence Controller state vector.
   Signal SEQ_STATE_N: std_logic_vector(1 downto 0);  -- Seq Controller next state vector.

-- Output signals of Remainder Selection Logic, 16
   Signal RMDR_LD: std_logic;     -- Remainder Reg load-enable signal.
   Signal RMDR_SEL : std_logic_vector( 5 downto 0); -- 6 bit vector of signals used
        -- to select the true remainder from the plurality of remainders.
        -- The vector is "one hot" encoded.
   Signal RMDR_VALID: std_logic;  -- Tags the contents of the Remainder Reg as valid.

--Signals to select subsets of TVn values to apply to adder inputs "A".
   Signal SEL_SET_1 : std_logic;  -- Signal to apply TV1, TV2, TV3 and TV4.
   Signal SEL_SET_2 : std_logic;  -- Signal to apply TV5, TV6, TV7 and TV8.
   Signal SEL_SET_3 : std_logic;  -- Signal to apply TV9, TV10, TV11 and TV12.

-- Adder 0 was reduced out of the design and thus eliminated, being replaced with
--    a straight-through bus, leaving the other 6 adders physically implemented.
--  Create a vector from the adder CY_BR outputs so they can be referenced as a group
--  in the CASE statement in the RMDR_SEL_LOGIC process, below.
   CY_BR_VEC(0) <= CY_BR1; -- CY_BR output of adder 1 is bit 0 of CY_BR_VEC.
   CY_BR_VEC(1) <= CY_BR2; -- CY_BR output of adder 2 is bit 1 of CY_BR_VEC.
   CY_BR_VEC(2) <= CY_BR3; -- CY_BR output of adder 3 is bit 2 of CY_BR_VEC.
   CY_BR_VEC(3) <= CY_BR4; -- CY_BR output of adder 4 is bit 3 of CY_BR_VEC.
```

FIGURE 7B

```
-- ***********   Behavior of Sequence Controller State Machine, 25 *************
SEQUENCE_CONTROLLER : process (RMDR_VALID, RESET) -- RMDR_VALID and RESET are input
                                                   -- signals.
-- The RMDR_VALID signal is generated in the RMDR_SEL_LOGIC process, below.

-- Sequence Controller state names and values.  Values are arbitrary.
Constant CYCLE_1 : std_logic_vector(1 downto 0) := "01";  -- Controller CYCLE_1 state.
Constant CYCLE_2 : std_logic_vector(1 downto 0) := "10";  -- Controller CYCLE_2 state.
Constant CYCLE_3 : std_logic_vector(1 downto 0) := "11";  -- Controller CYCLE_3 state.

-- Sequencer behavior
   If RESET = '1' then
      SEQ_STATE   <= CYCLE_1;  -- Reset to CYCLE_1 state.
   else  -- Normal sequence controller operation.
      Case SEQ_STATE is
         WHEN CYCLE_1 then
            If RMDR_VALID = '1' then  -- The true remainder is in this set.
               SEQ_STATE_N  <= CYCLE_1; -- Stay in CYCLE_1 state.
            else                  -- True remainder is not in this set.
               SEQ_STATE_N  <= CYCLE_2; -- Continue on to CYCLE_2 state.
            End if;
         WHEN CYCLE_2 then
            If RMDR_VALID = '1' then  -- The true remainder is in this set.
               SEQ_STATE_N  <= CYCLE_1; -- Go back to CYCLE_1 state.
            else                  -- True remainder is not in this set.
               SEQ_STATE_N  <= CYCLE_3; -- Continue on to CYCLE_3 state.
            End if;
         WHEN CYCLE_3 then  -- True remainder MUST be in this set if not found so far.
               SEQ_STATE_N  <= CYCLE_1; -- Return to CYCLE_1 state.
      End Case;
End SEQUENCE_CONTROLLER process;
```

FIGURE 7C

```
-- ***********   Behavior of Remainder Selection Logic, 16 *************

-- Remainder Select control output to control Remainder Mux, item # 19.
RMDR_SEL_LOGIC : process (SEQ_STATE,CY_BR_VEC)-- SEQ_STATE and CY_BR_VEC are input
                                              -- signals.
Note that RMDR_VALID signal output here is an input to the Sequencer Controller, above.

-- RMDR_SEL output signal names and values.  "One-hot" encoded.
Constant NONE : std_logic_vector(4 downto 0) := "00000";  -- Don't select any outputs.
Constant ZERO : std_logic_vector(4 downto 0) := "00001";  -- Select "adder 0" output.
Constant ONE  : std_logic_vector(4 downto 0) := "00010";  -- Select adder 1 output.
Constant TWO  : std_logic_vector(4 downto 0) := "00100";  -- Select adder 2 output.
Constant THREE: std_logic_vector(4 downto 0) := "01000";  -- Select adder 3 output.
Constant FOUR : std_logic_vector(4 downto 0) := "10000";  -- Select adder 4 output.

-- CY_BR_VEC names and values which match associated numerical quotient values.
Constant QUOT_A : std_logic_vector(3 downto 0) := "0000";  -- Quotient value is 0,4,8,12.
Constant QUOT_B : std_logic_vector(3 downto 0) := "0001";  -- Quotient value is 1,5 or 9.
Constant QUOT_C : std_logic_vector(3 downto 0) := "0011";  -- Quotient value is 2,6 or 10.
Constant QUOT_D : std_logic_vector(3 downto 0) := "0111";  -- Quotient value is 3,7 or 11.
Constant QUOT_E : std_logic_vector(3 downto 0) := "1111";  -- Quotient value is unknown.

Case SEQ_STATE is                 -- Each value of SEQ_STATE selects a different set of
                                  -- combinational logic to be performed.
  When CYCLE_1 then -- Perform sequencer cycle_1 logic function.
    SEL_SET_1 <= '1';  -- Apply TV1, TV2, TV3 and TV4 to adders.
    Case CY_BR_VEC is  -- Look at the CY_BR adder outputs.
      When QUOT_A =>   -- Quotient of dividend is 0.  This is the only state in which
                       -- the "adder 0" output is considered.
        RMDR_SEL     <= ZERO;    -- ."Adder 0" has correct mod output value.
        QUOTIENT_VAL <= "0000";  -- Quotient value is 0 (binary).
        RMDR_VALID   <= '1';     -- Remainder is valid.
        RMDR_LD      <= '1';     -- Load valid remainder into remainder reg, 23.
      When QUOT_B =>   -- Quotient of dividend is 1.
        RMDR_SEL     <= ONE;     -- Adder 1 has correct remainder output value.
        QUOTIENT_VAL <= "0001";  -- Quotient value is 1 (binary).
        RMDR_VALID   <= '1';     -- Remainder is valid.
        RMDR_LD      <= '1';     -- Load valid remainder into remainder reg, 23.
      When QUOT_C =>   -- Quotient of dividend is 2.
        RMDR_SEL     <= TWO;     -- Adder 2 has correct remainder output value.
        QUOTIENT_VAL <= "0010";  -- Quotient value is 2 (binary).
        RMDR_VALID   <= '1';     -- Remainder is valid.
        RMDR_LD      <= '1';     -- Load valid remainder into remainder reg, 23.
      When QUOT_D =>   -- Quotient of dividend is 3.
        RMDR_SEL     <= THREE;   -- Adder 3 has correct remainder output value.
        QUOTIENT_VAL <= "0011";  -- Quotient value is 3 (binary).
        RMDR_VALID   <= '1';     -- Remainder is valid.
        RMDR_LD      <= '1';     -- Load valid remainder into remainder reg, 23.
      When QUOT_E =>   -- Quotient of dividend might be 4.
        RMDR_SEL     <= FOUR;    -- Adder 4 may have correct remainder output value.
        RMDR_VALID   <= '0';     -- Remainder is unknown.  Sequencer must continue.
        RMDR_LD      <= '1';     -- Load remainder into remainder reg, 23. It might
                                 -- be valid.  Must be tested next cycle.
    End case;
```

FIGURE 7D

```
When CYCLE_2 then -- Perform sequencer cycle_2 logic function.
   SEL_SET_2 <= '1';  -- Apply TV5, TV6, TV7 and TV8 to adders.
   Case CY_BR_VEC is  -- Look at the CY_BR adder outputs.
      When QUOT_A =>   -- Quotient of dividend is 4.  This confirms the possibility.
         RMDR_SEL      <= NONE;     -- .All adder outputs are incorrect.
         QUOTIENT_VAL <= "0100";   -- Quotient value is 4 (binary).
         RMDR_VALID    <= '1';      -- Remainder in Remainder Reg, 23, is valid.
         RMDR_LD       <= '0';      -- Don't load.  Hold valid remainder previously
                                    -- loaded into remainder reg, 23, in CYCLE_1.
      When QUOT_B =>   -- Quotient of dividend is 5.
         RMDR_SEL      <= ONE;      -- Adder 1 has correct remainder output value.
         QUOTIENT_VAL <= "0101";   -- Quotient value is 5 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_C =>   -- Quotient of dividend is 6.
         RMDR_SEL      <= TWO;      -- Adder 2 has correct remainder output value.
         QUOTIENT_VAL <= "0110";   -- Quotient value is 6 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_D =>   -- Quotient of dividend is 7.
         RMDR_SEL      <= THREE;    -- Adder 3 has correct remainder output value.
         QUOTIENT_VAL <= "0111";   -- Quotient value is 7 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_E =>   -- Quotient of dividend might be 8.
         RMDR_SEL      <= FOUR;     -- Adder 4 may have correct remainder output value.
         RMDR_VALID    <= '0';      -- Remainder is unknown.  Sequencer must continue.
         RMDR_LD       <= '1';      -- Load remainder into remainder reg, 23. It might
                                    -- be valid.  Must be tested next cycle.
   End case;
When CYCLE_3 then -- Perform sequencer cycle_3 logic function.
   SEL_SET_3 <= '1';  -- Apply TV9, TV10, TV11 and TV12 to adders.
   Case CY_BR_VEC is  -- Look at the CY_BR adder outputs.
      When QUOT_A =>   -- Quotient of dividend is 8.  This confirms the possibility.
         RMDR_SEL      <= NONE;     -- .All adder outputs are incorrect.
         QUOTIENT_VAL <= "1000";   -- Quotient value is 8 (binary).
         RMDR_VALID    <= '1';      -- Remainder in Remainder Reg, 23, is valid.
         RMDR_LD       <= '0';      -- Don't load.  Hold valid remainder previously
                                    -- loaded into remainder reg, 23, in CYCLE_1.
      When QUOT_B =>   -- Quotient of dividend is 9.
         RMDR_SEL      <= ONE;      -- Adder 1 has correct remainder output value.
         QUOTIENT_VAL <= "1001";   -- Quotient value is 9 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_C =>   -- Quotient of dividend is 10.
         RMDR_SEL      <= TWO;      -- Adder 2 has correct remainder output value.
         QUOTIENT_VAL <= "1010";   -- Quotient value is 10 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_D =>   -- Quotient of dividend is 11.
         RMDR_SEL      <= THREE;    -- Adder 3 has correct remainder output value.
         QUOTIENT_VAL <= "1011";   -- Quotient value is 11 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
      When QUOT_E =>   -- Quotient of dividend is 12.
         RMDR_SEL      <= FOUR;     -- Adder 4 has correct remainder output value.
         QUOTIENT_VAL <= "1100";   -- Quotient value is 12 (binary).
         RMDR_VALID    <= '1';      -- Remainder is valid.
         RMDR_LD       <= '1';      -- Load valid remainder into remainder reg, 23.
   End case;
End case;
```